ELSEVIER

Artificial Intelligence 134 (2002) 101–120

Artificial Intelligence

www.elsevier.com/locate/artint

A hierarchical approach to computer Hex

Vadim V. Anshelevich

1200 Navaho Trail, Richardson, TX 75080, USA

Abstract

Hex is a beautiful game with simple rules and a strategic complexity comparable to that of Chess and Go. The massive game-tree search techniques developed mostly for Chess and successfully used for Checkers and a number of other games, become less useful for games with large branching factors like Hex and Go. In this paper, we describe deduction rules, which are used to calculate values of complex Hex positions recursively starting from the simplest ones. We explain how this approach is implemented in HEXY—the strongest Hex-playing computer program, the Gold medallist of the 5th Computer Olympiad in London, August 2000.

Keywords: Hex; Game programming; Theorem proving

1. Hex and its history

The game of Hex was invented by the Danish poet and mathematician P. Hein. He introduced the game in 1942 in a lecture to students at the Niels Bohr Institute for Theoretical Physics. The game soon became popular in Denmark under the name of Polygon. It was independently reinvented by J. Nash in 1948 when he was a graduate student at Princeton University. Parker Brothers marketed a version of the game in 1952 under the name Hex. The game was presented to the general public by M. Gardner in *Scientific American* (see [12,13]).

Hex is a two-player game played on a rhombic board with hexagonal cells (see Fig. 1). The classic board is 11×11, but it can be any size. The 10×10, 14×14 and even 19×19 board sizes are also popular. The players, Black and White, take turns placing pieces of their color on empty cells of the board. Black's objective is to connect the two opposite black sides of the board with a chain of black pieces. White's objective is to connect the two opposite white sides of the board with a chain of white pieces (see Fig. 1). The player moving first has a big advantage in Hex. In order to equalize chances, players often employ

E-mail address: vanshel@earthlink.net (V.V. Anshelevich).

0004-3702/01/$ – see front matter
PII: S0004-3702(01)00154-0

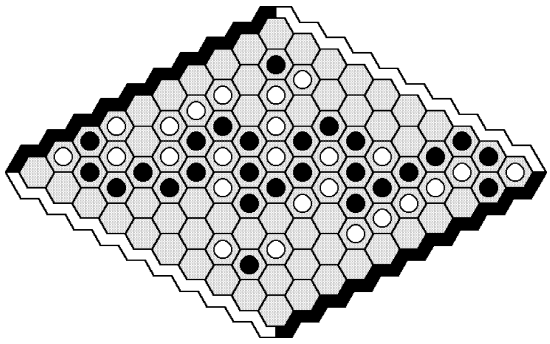

Fig. 1. The chain of black pieces connects black boundaries. Black has won the game.

a "swap" rule where the second player has the option of taking the first player's opening move. Despite the simplicity of the rules, the game's strategic and tactical ideas are rich and subtle. An introduction to Hex strategy and tactics can be found in the book by Browne [7].

The game can never end in a draw. This follows from the fact that if all cells of the board are occupied, then a winning chain for Black or White must exist. While this two-dimensional topological fact seems obvious, it is not at all trivial. In fact, Gale [11] demonstrated that this result is equivalent to the Brouwer fixed-point theorem for 2-dimensional squares. It follows that there exists a winning strategy either for the first or second player. Using a "strategy stealing" argument (cf. [5]), Nash showed that a winning strategy exists for the first player. However, this is only a proof of existence, and it does not provide any clues that can help the first player to win. One can find many examples of opening positions on small boards up to 6×6 (including all one-stone positions) with indications of which player has a winning strategy at the web site: http://www.cs.ualberta.ca/∼queenbee/openings.html.

Even and Tarjan [10] showed that the problem of determining which player has a winning strategy in a generalization of Hex, called the Shannon switching game on vertices, is PSPACE complete. Reisch [17] later proved this for $N \times N$ Hex itself.

A Hex-playing machine was built by Shannon and Moore [20]. Shannon associated a two-dimensional electrical charge distribution with any given Hex position. This machine made decisions based on properties of the corresponding potential field. We acknowledge that our work is greatly inspired by the beauty of Shannon's original idea.

Although the rules of Hex are simple, the game requires deep strategic understanding and sharp tactical skills. The massive game-tree search techniques developed over the last 30 to 40 years mostly for Chess (see [1,8]), and successfully used for Checkers [19] and a number of other games, become less useful for games with large branching factors like Hex and Go. For a classic 11×11 Hex board, the average number of legal moves is about 100 (cf. 38 for Chess [14] and 8 for Checkers [19]).

Combinatorial Game Theory provides very powerful tools for the analysis of sums of large numbers of relatively simple games (see [5,9]). It is also useful in situations where complex positions can be decomposed into sums of simpler ones. In particular, this method

is applicable to an analysis of Go endgames [6,16]. Although Hex positions do not tend to decompose into sums of local sub-games, many Hex positions can still be considered as combinations of simpler sub-games.

In this article, we concentrate on the hierarchy of Hex sub-games, define a set of deduction rules, and demonstrate how these deduction rules can be used to calculate values of complex sub-games recursively, starting from the simplest ones. Integrating the information about sub-games of this hierarchy, we build an evaluation function foreseeing the potential of Hex positions many moves ahead. This approach is implemented in HEXY, currently the strongest Hex-playing computer program. It won the gold medal of the 5th Computer Olympiad in London, August 2000. HEXY does not perform massive game-tree search. Instead, the program focuses on a deep analysis of the sub-games hierarchy for a relatively small number of game positions. A Windows version of the program is publicly available at http://home.earthlink.net/∼vanshel.

The course of the article is as follows. Section 2 discusses the concept of virtual connections. In Section 3 we describe the AND and OR deduction rules, and in Section 4 we define the H-search process for the hierarchical calculation of virtual connections. Subsequently, Section 5 presents an electrical resistor circuit model, which is used to combine information about the hierarchy of virtual connections into a global evaluation function. In Section 6 we explain how this approach is implemented in HEXY. The major ideas of this work were presented earlier in [2,3].

2. Virtual connections and virtual semi-connections

In Sections 2, 3, and 4 we characterize Hex positions from Black's point of view. White's point of view can be considered in a similar way. We consider the four polygonal boundary bands as additional cells (see Fig. 2) and assume that the black boundary cells are permanently occupied by black pieces, and the white boundary cells are permanently occupied by white pieces.

Consider the two diagrams in Fig. 2. In both positions White cannot prevent Black from connecting the two groups of black pieces, x and y, even if White moves first, since there are two empty cells a and b adjacent to both x and y. If White occupies one of those empty cells, then Black can play the other. Note that the black connection between groups x and y is secured as long as the two cells a and b remain empty. In this type of position we say that the groups of black pieces x and y form a *two-bridge*. In a battle, where Black tries

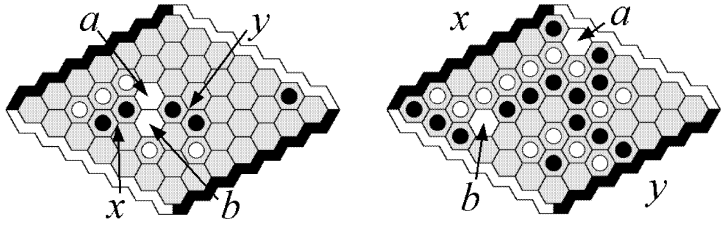

Fig. 2. The groups of black pieces (x and y) form a two-bridge.

to connect the groups x and y, and White tries to prevent it, the result is predictable two moves ahead. This provides an important advantage to Black. In the position on the left, this advantage is local. In the position on the right this advantage is decisive, the groups are connected to the black boundaries, and White should resign.

Before generalizing the two-bridge concept, we introduce two notions. First, we define a cell to be black if it is occupied by a black piece; second, we refer to a group of connected black cells as a single black cell. The following three definitions generalize the two-bridge concept.

Definition 1. Let x and y be two different cells, and let A be a set of empty cells of a given position. We assume that $x \notin A$ and $y \notin A$. Consider the triplet (x, A, y) as a *sub-game*, where Black tries to connect cells x and y with a chain of black pieces, White tries to prevent it, and both players can put their pieces only in cells of A. We then define x and y as *ends* of the sub-game, and A as its *carrier*.

Definition 2. A sub-game is a *virtual connection* iff Black has a winning strategy even if White moves first.

Definition 3. A sub-game is a *virtual semi-connection* iff Black has a winning strategy moving first, and does not have one if he moves second.

We represent virtual connections and virtual semi-connections with diagrams as in Fig. 3. In both cases we see black-black, black-empty, and empty-empty ends. The blank rectangles stand for carriers of virtual connections. The crossed rectangles stand for carriers of virtual semi-connections.

In practice, it is more convenient to use the following recursive definitions.

Definition 2a. A sub-game is a virtual connection iff for every white move there exists a black move such that the resulting sub-game is a virtual connection.

Definition 3a. A sub-game is a virtual semi-connection iff it is not a virtual connection, and there exists a black move such that the resulting sub-game is a virtual connection.

Assume that in a given position with a virtual connection, White moves first. The number of moves which must be made in order for Black to win this sub-game, under the condition that Black tries to minimize this number and White tries to maximize it, characterizes the *depth of the virtual connection*. In other words, the depth of the virtual connection

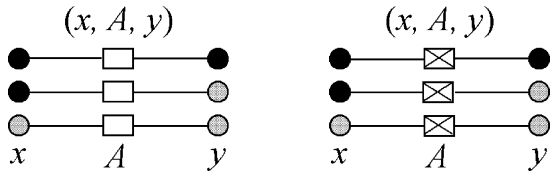

Fig. 3. Diagrams of virtual connections (left) and virtual semi-connections (right).

is the depth of a game-tree search required to establish this virtual connection. Virtual connections with depth d contain information about the nodes of the game tree associated with the Hex position d moves ahead.

Four remarks are in order:

- A pair of neighboring cells forms a virtual connection with an empty carrier. The depth of this virtual connection is equal to zero.
- A two-bridge forms a virtual connection of depth two.
- The ends x and y can form virtual connections with several different carriers. The virtual connection (x, A, y) is *minimal* iff there does not exist a virtual connection (x, B, y) such that $B \subset A$ and $B \neq A$. If a sub-game (x, A, y) is a virtual connection and (x, C, y) is a sub-game such that $A \subset C$, then the sub-game (x, C, y) is also a virtual connection. This is why we are primarily interested in minimal virtual connections.
- A special role is played by a *winning virtual connection* formed by the additional boundary cells. If it exists, then there exists a global winning strategy for Black even if White moves first.

In Figs. 4 and 5 one can see four typical samples of virtual connections and virtual semi-connections.

In each diagram of Fig. 4 the cell y is formed by the black piece connected to the bottom right black boundary. The cells of their carriers are marked white. The characterization is as follows.

(1) An "edge connection" from the fourth row. Depth = 10.
(2) A "ladder". Depth = 14.
(3) A chain of two-bridges. Depth = 12.
(4) This virtual connection will be analyzed in Section 3. Depth = 6 (see Fig. 8).

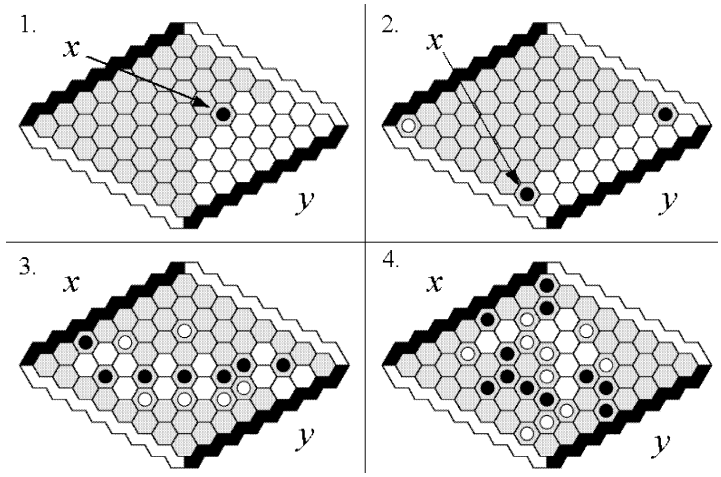

Fig. 4. Black cells x and y form virtual connections.

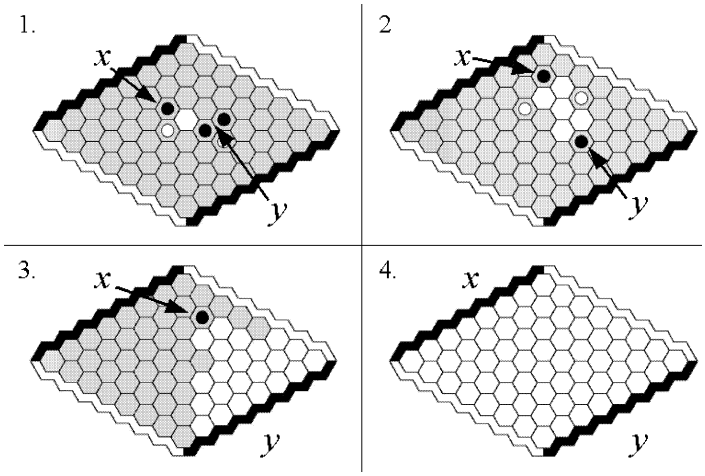

Fig. 5. Black cells x and y form virtual semi-connections.

In the four diagrams of Fig. 5 we see four samples of virtual semi-connections. The cells of their carriers are marked white. Diagram 4 shows the initial position. According to Nash's theorem mentioned in Section 1, the initial position is a virtual semi-connection.

3. Deduction rules

In this section we define two deduction rules, which can be used to build complex virtual connections starting from the simplest ones.

The AND Deduction Rule. *Let sub-games* (x, A, u) *and* (u, B, y) *be two virtual connections, with common end* u *and different ends* $x \neq y$. *We assume that* $x \notin B$, $y \notin A$, *and* $A \cap B = \emptyset$.

(i) *If* u *is black, then the sub-game* $(x, A \cup B, y)$ *is a virtual connection.*

(ii) *If* u *is empty, then the sub-game* $(x, A \cup u \cup B, y)$ *is a virtual semi-connection.*

Proof. (i) Since $A \cap B = \emptyset$, White cannot attack both virtual connections simultaneously. Assume that White occupies a cell $a \in A$. Since the sub-game (x, A, u) is a virtual connection, there exists a cell $b \in A$, where Black can play to create a new virtual connection (x, A', u). The new carrier A' is obtained from A by removing two cells a and b. (Note that the new virtual connection belongs to a position different from the original one). In short, if White occupies a cell in A, then Black can restore the first virtual connection by moving to an appropriate cell of A. The same is true for B, and thus the result follows by induction.

(ii) If the cell u is empty, then Black can occupy this cell, and case (ii) is reduced to case (i). □

Fig. 6 shows a graphical representation of this deduction rule.

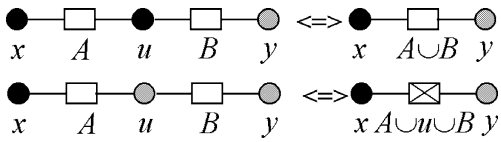

Fig. 6. The AND Deduction Rule.

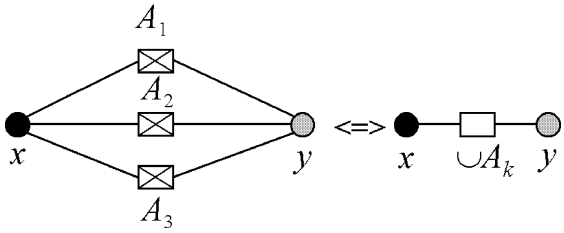

Fig. 7. The OR Deduction Rule.

The OR Deduction Rule. *Let sub-games* (x, A_k, y) $(k = 1, 2, \ldots, n,$ *for* $n > 1)$ *with common ends* x *and* y *be virtual semi-connections. If*

$$\bigcap_{k=1}^{n} A_k = \emptyset,$$

then the sub-game (x, A, y)*, where*

$$A = \bigcup_{k=1}^{n} A_k,$$

is a virtual connection.

Proof. If White occupies a cell $a \in A_i$, then there exists a different carrier A_j, such that $a \notin A_j$, and Black can move to A_j to convert virtual semi-connection (x, A_j, y) to a virtual connection. □

Fig. 7 graphically represents this deduction rule for $n = 3$.

Fig. 8 demonstrates how the AND and OR deduction rules can be used for proving virtual connections. Diagram 1 of Fig. 8 represents the sub-game on the board. The sequence of transformations in Diagrams 2 through 6 graphically demonstrates the application of the AND and OR deduction rules, and proves that Black has a winning position, even if White moves first.

Diagram 3 is obtained from Diagram 1 by applying the AND Deduction Rule six times and then the OR Deduction Rule three times. Diagram 4 results from the AND Deduction Rule. The winning virtual connection in Diagram 6 follows from applying the AND Deduction Rule two times and a final application of the OR Deduction Rule.

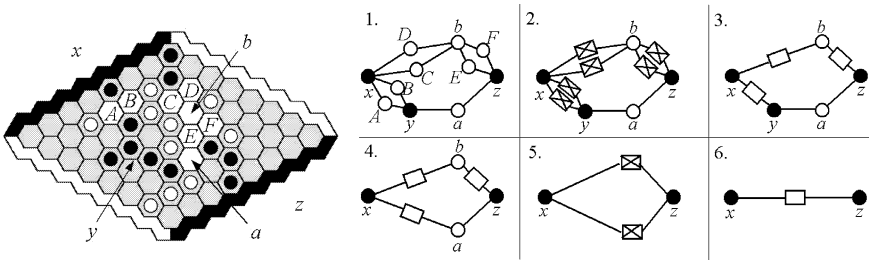

Fig. 8. The use of AND and OR deduction rules.

4. H-search

The AND and OR deduction rules introduced in Section 3 are used to define the following hierarchical algorithm for the calculation of virtual connections.

H-search. Consider some initial set of virtual connections and semi-connections as the first generation of virtual connections and semi-connections. In particular, this initial set might consist of the simplest virtual connections, namely all pairs of neighboring cells. By applying the AND and OR deduction rules to the appropriate groups of the first generation of virtual connections and semi-connections, we build the second generation of virtual connections and semi-connections. Then we apply the AND and OR deduction rules to both the first and the second generations of virtual connections and semi-connections to build the third generation of virtual connections and semi-connections, etc. This process stops when either no new virtual connections are produced or the winning virtual connection is built.

We consider that to some extent this process is analogous to automatic theorem proving (see for example [18]). Appendix A contains a skeleton of an algorithm representing the H-search for calculation of virtual connections.

H-search can build all of the virtual connections shown in Figs. 2 and 4. Section 3 explained the case of the sub-game of Diagram 4 in Fig. 4. Formal proofs for the sub-games of Diagrams 1 and 2 in Fig. 4 are given in Appendix B. In Appendix C we analyze how H-search works in case of two simple examples of virtual connections containing a large parameter.

We emphasize that H-search is not another version of the usual game-tree search. H-search and game-tree search use different representations of the problem and, as a result, they search in different spaces. In particular, H-search performs a search for virtual connections in the set of sub-games of a Hex position.

The question now arises: is the set of the AND and OR deduction rules *complete*, i.e., can H-search build *all* virtual connections starting from the simplest ones? The answer is no. Fig. 9 shows a counter-example. It represents a virtual connection that cannot be built by H-search. It is easy to check that this sub-game is a virtual connection. Indeed, if White plays at a, Black can reply with b, forcing White to occupy c. Then Black plays d securing the win. This virtual connection is formed by two equivalent virtual semi-

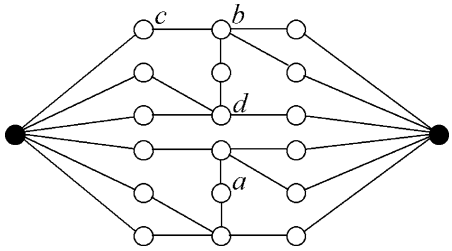

Fig. 9. The two black cells form a virtual connection, which cannot be built using the AND and OR deduction rules.

connections with disjoint carriers connected in parallel. A computer program was used to verify that no combination of the AND and OR deduction rules can establish these virtual semi-connections or the overall virtual connection.

In addition to the above, we remark that the AND and OR deductions rules can be generalized in a way that makes them complete [2]. However, to the best of our knowledge, generalized H-search has not been implemented so far, and its practical value is not clear.

5. Evaluation function and electrical resistor circuits

In practice, due to limited computing resources, both H-search and game-tree search are capable of discovering a winning virtual connection either on small boards or in endgames. A game-tree search for a winning strategy (a winning virtual connection in the case of Hex) is usually replaced by the search for the minimax value of some heuristic function on a smaller graph, where computations are still feasible. In the case of H-search, the situation is similar. Even if it is impossible to build the winning virtual connection for Black or for White due to incompleteness of the AND and OR deduction rules and/or the limited computing resources, the set of discovered virtual connections provides useful information for the evaluation of the entire position. In this section we introduce a family of evaluation functions based on an *electrical resistor circuit* representation of Hex positions. The evaluation functions combine information about the hierarchy of discovered virtual connections.

One can think of an electrical circuit as a graph. Edges of the graph play the role of electrical links (resistors). The resistance of each electrical link is equal to the length of the corresponding edge of the graph. Here, we see that the "electrical circuit" language better suits our needs. With every Hex position, we associate two electrical circuits. The first one characterizes the position from Black's point of view (Black's circuit), and the second one characterizes the position from White's point of view (White's circuit). To every cell c of the board we assign a resistance r in the following way.

For Black's circuit:

$$r_{\mathrm{B}}(c) = \begin{cases} 1 & \text{if } c \text{ is empty,} \\ 0 & \text{if } c \text{ is occupied by a black piece,} \\ +\infty & \text{if } c \text{ is occupied by a white piece.} \end{cases}$$

For White's circuit:

$$r_{\mathrm{W}}(c) = \begin{cases} 1 & \text{if } c \text{ is empty,} \\ 0 & \text{if } c \text{ is occupied by a white piece,} \\ +\infty & \text{if } c \text{ is occupied by a black piece.} \end{cases}$$

For each pair of neighboring cells, (c_1, c_2), we associate an electrical link with resistance:

$$r_{\mathrm{B}}(c_1, c_2) = r_{\mathrm{B}}(c_1) + r_{\mathrm{B}}(c_2) \quad \text{for Black's circuit,}$$

$$r_{\mathrm{W}}(c_1, c_2) = r_{\mathrm{W}}(c_1) + r_{\mathrm{W}}(c_2) \quad \text{for White's circuit.}$$

In the first instance, the circuits only take into account connections between neighboring cells. We will enhance these circuits by incorporating information about the hierarchy of discovered virtual connections. Below we focus on Black's circuits only. White's circuits can be dealt with in a similar way.

Our first approach was to add an additional electrical link to Black's circuit between two cells x and y if x and y form a virtual connection. Consequently, all virtual connections would be treated as neighboring cells. However, virtual connections between nearest neighbors are stronger than other virtual connections. So, our circuit should reflect this. Thus, we made the resistance between the ends of all other virtual connections higher than between two nearest neighbors.

In the second approach we added other links to Black's circuit in the following way. If two black cells x and y form a virtual connection and an empty cell c is a neighbor of one of the ends of the virtual connection, say x, we then treat cell c also as a neighbor of the other end y. This means that we connect cells c and y with an additional electrical link in the same way as the actual neighbors. (We have tried both approaches, but we do not have conclusive data to suggest which one is better.)

Let R_{B} and R_{W} be distances between black boundaries in Black's circuit and white boundaries in White's circuit, correspondingly. Now we define an evaluation function E as

$$E = \log(R_{\mathrm{B}}/R_{\mathrm{W}}).$$

A reasonable distance metric is the length of the shortest path on the graph connecting boundaries. However, distances can be measured in different ways. Following Shannon's idea, we applied an electrical voltage to the opposite boundaries of the board and measured the total resistance between them, R_{B} for Black's circuit and R_{W} for White's circuit (see Fig. 10). This idea was implemented by C.E. Shannon in a robot which played the game Bird Cage, also known as the game of Gale or Bridg-it (see [13]).

We prefer this method for measuring distances because according to the Kirchhoff electrical current laws, the total resistance takes into account not only the length of the shortest path, but also all other paths connecting the boundaries, their lengths, and their intersections. Yet, our choice for this measure is not as important as the depths of the virtual connections included in the circuit. Virtual connections with a depth d contain information about the nodes of the game tree associated with the evaluation of the Hex position d moves ahead. Thus, we can expect that by including electrical links that correspond to

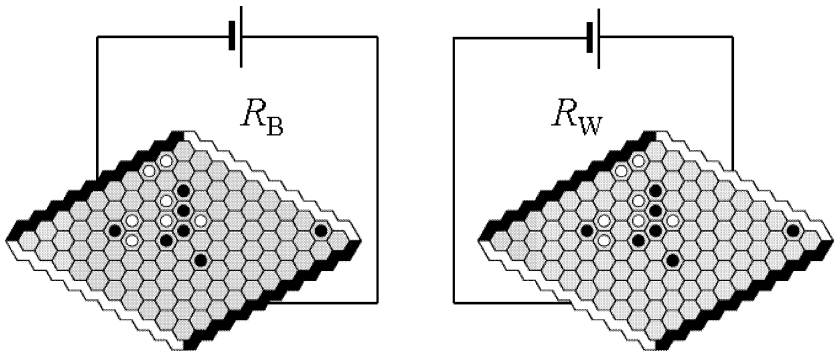

Fig. 10. Black's and White's circuits.

virtual connections with a depth less than or equal to d, we obtain an evaluation function with predicting abilities up to d moves ahead.

6. HEXY plays Hex

HEXY is a Hex-playing computer program, which uses the ideas presented above. It runs on a standard PC with Windows, and can be downloaded from the website: http://home.earthlink.net/∼vanshel.

As shown in Section 4, there exist virtual connections that cannot be discovered by H-search. This means that H-search cannot serve as an alternative to the game-tree search. Therefore, HEXY uses a combined algorithm, mixing both methods in the proportion described below.

HEXY uses a selective alpha-beta search algorithm with the evaluation functions described in Section 5. For every node to be evaluated HEXY calculates the hierarchy of the virtual connections for both Black's and White's circuits using H-search. Then HEXY calculates the evaluation function based on resistances R_B and R_W between Black and White boundaries, correspondingly. To calculate the electrical resistance between boundaries, HEXY solves the Kirchhoff system of linear equations for electric potential using a method of iterations (see, for example, [21]). The solution is used for move ordering too. For every empty cell of the board, the program calculates an energy, which dissipates on all links adjacent to this cell. Moves played in the cells with a higher energy have a higher priority.

In practice, HEXY rarely uses pairs of neighboring cells as the initial set for H-search, but looks for changes in the hierarchy of the virtual connections caused by an additional piece placed on the board. The program keeps track of the available minimal virtual connections and semi-connections.

The program has several important thresholds, which control the run time and the amount of memory required. Since the major objective was the creation of a Hex-playing program that provides fun for Hex fans, we imposed the condition that HEXY should be able to complete a game on the 10×10 board within 8 minutes on a standard PC with

300 MHz processor and 32 MB RAM. This means that Hexy should not spend more than 10 to 20 seconds per move. Thus, we tried to maximize the playing strength by finding optimal values of the thresholds, satisfying the above condition. This was done experimentally. During our experiments we compared the playing strength of different versions of the program applying the following methodology. We selected five first moves, which we considered as neutral, and played one version of the program against another in a 10-game match. Each starting position was played twice, allowing each program to enjoy the first move (no swap was allowed). If one of the programs won at least eight games out of ten, we decided that this program was superior. Otherwise, we considered the result of the match as being inconclusive.

The most important thresholds are D and M. The parameter D is the depth of the game-tree search. The parameter M sets a limit on the number of different minimal virtual connections with the same ends built by the program. This threshold indirectly controls the total number of calculated minimal virtual connections. The larger M is, the more minimal virtual connections H-search builds for every node of the game-tree. However, we do not put any limits on the number of iterations of the algorithm or on the total number of virtual connections and their depths or on the total processing time. H-search stops when either no new virtual connections are produced or the winning virtual connection has been built.

There is a trade-off between the parameters D and M, and finding an optimum is an important task. Experiments show that the dependence of HEXY's playing strength on the parameter M is much more significant than its dependence on the depth D of the game-tree search. The best results are obtained with $D = 3$ and $M = 20$ (for the 10×10 board). In particular, this version of HEXY (called Advanced Level) is significantly stronger than the one with $D = 6$ and $M = 0$ ($M = 0$ means that no virtual connections except pairs of neighboring cells are calculated). Advanced Level performs a shallow game-tree search (200–500 nodes per move), yet it calculates many dozens of virtual connections and many hundreds of virtual semi-connections per node. This version of HEXY routinely detects virtual connections with depth 20 or more, which means that the program foresees some lines of play 20 or more moves ahead. In particular, HEXY routinely discovers the winning virtual connection 20 or more moves prior to the actual end of the game. (The fact that the typical maximal depth of calculated virtual connections is approximately equal to 20 as well as the chosen value for M is a pure coincidence.)

Another threshold, K, sets the limit for the number of virtual semi-connections on the input side of the OR deduction rule. It is selected in the range 4 to 5. No other H-search pruning or filtering is applied.

HEXY demonstrates a clear superiority over all known Hex-playing computer programs. In particular, we compare HEXY with the program QUEENBEE by J. van Rijswijck (University of Alberta GAMES group). In contrast to HEXY, this program exploits a massive game-tree search, with a computationally inexpensive evaluation function (see [22]). QUEENBEE uses the iterative-deepening alpha-beta search enhanced with Minimal Window/Principal Variation Search and transposition tables. QUEENBEE's search also incorporates the fractional-ply searching ideas of the "Sex Search algorithm" [15]. The comparison of HEXY with QUEENBEE was performed for the boards 7×7, 10×10 and 11×11, using the same methodology and criteria as the ones used for comparison

of different versions of HEXY. On all these boards, HEXY demonstrated superiority. The programs also faced each other in the Hex tournament of the 5th Computer Olympiad in London, 2000. HEXY defeated QUEENBEE with four wins, no losses [4]. (According to the Olympiad rules, each program had 30 minutes for a game on the 11×11 board. The parameters of HEXY were adjusted in order to meet these requirements. Since HEXY did not use any opening book, 40 percent of the available time was allocated for the first four moves.)

HEXY has also been tested on the 10×10 board against human players on the popular game website Playsite (http://www.playsite.com/games/board/hex). The provisional conclusion is that HEXY cannot compete on an equal footing with the best human players. Nevertheless, after more than 100 games, the program achieved a rating that is within the highest Playsite rating range.

7. Conclusion

In this paper we described a hierarchical approach to the game of Hex and explained how this approach is implemented in HEXY, currently the strongest Hex-playing computer program. We have concentrated on the hierarchy of virtual connections, and have defined the AND and OR deduction rules used to build complex virtual connections recursively starting from the simplest ones. Integrating the information about virtual connections of this hierarchy, we built an evaluation function capable of predicting the potential of Hex positions many moves ahead. The process of discovering virtual connections, H-search, is computationally expensive. Nevertheless, the resulting far-sighted abilities of the evaluation function greatly outweigh its computational cost.

Appendix A

Below we present a skeleton of a brute-force algorithm, which implements H-search for the calculation of Black's virtual connections (VC) and virtual semi-connections (VSC). The algorithm deals with the following data structures:

- G—a list of black and empty cells of the board. A group of connected black cells is considered as a single cell.
- C(g1, g2)—a list of carriers of VCs with ends g1 and g2 for every g1, g2 $\in$ G.
- SC(g1, g2)—a list of carriers of VSCs with ends g1 and g2 for every g1, g2 $\in$ G.

If we start from the simplest virtual connections, then initially these lists contain only nearest neighbors:

- C(g1, g2) $= \emptyset$, if g1 and g2 are not nearest neighbors,
- C(g1, g2) contains just empty carrier $\emptyset$, if g1 and g2 are nearest neighbors.
- SC(g1, g2) $= \emptyset$ for all g1 and g2.

In general, we can start from any lists of VCs and VSCs. This is useful when we look for changes in these lists caused by an additional piece placed on the board.

The procedure **H-SEARCH** recursively calculates VCs and VSCs starting from the ones present in the lists C(g1, g2) and SC(g1, g2), and updates all of these lists at each step of

the algorithm. We consider a VC as new one if it has been produced at the current or at the previous step of the algorithm.

PROCEDURE: H-SEARCH

```
WHILE (there is at least one new VC):
  LOOP1 over g ∈ G
    LOOP2 over g1, g2 ∈ G such that:
          g1 ≠ g2,
          at least one of the lists C(g1, g) or C(g2, g) contains at least one new carrier.
          If g is black then, additionally, g1 and g2 should be both empty.
      LOOP3 over c1 ∈ C(g1, g) and c2 ∈ C(g2, g) such that:
            At least one of the carriers c1 or c2 is new,
            c1 ∩ c2 = ∅,
            g1 ∉ c2 and g2 ∉ c1.
        IF (g is black)
          c = c1 ∪ c2.                    // the AND Deduction Rule
          UPDATE C(g1, g2) with c.
        ELSE
          sc = c1 ∪ g ∪ c2.               // the AND Deduction Rule
          UPDATE SC(g1, g2) with sc.
          IF (the last UPDATE is successful)
            APPLY_THE_OR_DEDUCTION_RULE_AND_UPDATE
              (C_SET = C(g1, g2),
               SC_SET = SC(g1, g2) − sc,
               UNION = sc,
               INTERSECTION = sc )
          END of IF
        END of IF
      END of LOOP3
    END of LOOP2
  END of LOOP1
END of WHILE
END of PROCEDURE
```

The recursive procedure **APPLY_THE_OR_DEDUCTION_RULE_AND_UPDATE** has four parameters: **C_SET, SC_SET**, **UNION** and **INTERSECTION**.

PROCEDURE: APPLY_THE_OR_DEDUCTION_RULE_AND_UPDATE
(C_SET = C, **SC_SET** = SC, **UNION** = u, **INTERSECTION** = i)

```
LOOP over sc1 ∈ CS
  u1 = u ∪ sc1,
  i1 = i ∩ sc1.
  IF (i1 = ∅)                  // the OR Deduction Rule
    UPDATE C with u1
```

```
    ELSE
      APPLY_THE_OR_DEDUCTION_RULE_AND_UPDATE (C_SET = C,
        SC_SET = SC − sc1, UNION = u1, INTERSECTION = i1 )
    END of IF
  END of LOOP
END of PROCEDURE
```

Appendix B

Below we show how to prove the virtual connections of Fig. B.1 (cf. Fig. 4, Diagrams 1 and 2) using the AND and OR deduction rules. In each diagram the cell y is formed by the black boundary. The cells of the carriers are enumerated. Diagram 1 is an "edge connection" from the fourth row and Diagram 2 is a "ladder".

We use abbreviation VC for virtual connection, VSC for virtual semi-connections, square brackets [] for carriers, and parentheses () for sub-game triplets.

Examples.

$[a, b]$ is a carrier consisting of two cells a and b.
[] is an empty carrier.
$(x, [a, b, c, d], y)$ is a sub-game with ends x and y and carrier $[a, b, c, d]$.
$(x, [], y)$ is a sub-game with ends x and y and an empty carrier.

The "edge connection" from the fourth row

The following sequence of deductions proves this virtual connection.

(x,[],16) is VC, (16,[],15) is VC. Apply AND: (x,[16],15) is VSC.
(x,[],19) is VC, (19,[],15) is VC. Apply AND: (x,[19],15) is VSC.
(x,[16],15) is VSC, (x,[19],15) is VSC. Apply OR: (x,[16,19],15) is VC.

(15,[],9) is VC, (9,[],8) is VC. Apply AND: (15,[9],8) is VSC.
(15,[],14) is VC, (14,[],8) is VC. Apply AND: (15,[14],8) is VSC.
(15,[9],8) is VSC, (15,[14],8) is VSC. Apply OR: (15,[9,14],8) is VC.

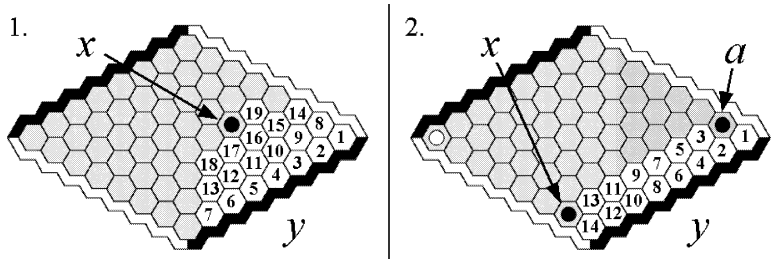

Fig. B.1. Black cells x and y form virtual connections.

(8,[],1) is VC, (1,[],y) is VC. Apply AND: (8,[1],y) is VSC.
(8,[],2) is VC, (2,[],y) is VC. Apply AND: (8,[2],y) is VSC.
(8,[1],y) is VSC, (8,[2],y) is VSC. Apply OR: (8,[1,2],y) is VC.

(10,[],3) is VC, (3,[],y) is VC. Apply AND: (10,[3],y) is VSC.
(10,[],4) is VC, (4,[],y) is VC. Apply AND: (10,[4],y) is VSC.
(10,[3],y) is VSC, (10,[4],y) is VSC. Apply OR: (10,[3,4],y) is VC.

(15,[],10) is VC, (10,[3,4],y) is VC. Apply AND: (15,[3,4,10],y) is VSC.
(15,[9,14],8) is VC, (8,[1,2],y) is VC. Apply AND: (15,[1,2,8,9,14],y) is VSC.
(15,[3,4,10],y) is VSC, (15,[1,2,8,9,14],y) is VSC.
Apply OR: (15,[1,2,3,4,8,9,10,14],y) is VC.

(11,[],4) is VC, (4,[],y) is VC. Apply AND: (11,[4],y) is VSC.
(11,[],5) is VC, (5,[],y) is VC. Apply AND: (11,[5],y) is VSC.
(11,[4],y) is VSC, (11,[5],y) is VSC. Apply OR: (11,[4,5],y) is VC.

(13,[],6) is VC, (6,[],y) is VC. Apply AND: (13,[6],y) is VSC.
(13,[],7) is VC, (7,[],y) is VC. Apply AND: (13,[7],y) is VSC.
(13,[6],y) is VSC, (13,[7],y) is VSC. Apply OR: (13,[6,7],y) is VC.

(17,[],12) is VC, (12,[],13) is VC. Apply AND: (17,[12],13) is VSC.
(17,[],18) is VC, (18,[],13) is VC. Apply AND: (17,[18],13) is VSC.
(17,[12],13) is VSC, (17,[18],13) is VSC. Apply OR: (17,[12,18],13) is VC.

(17,[],11) is VC, (11,[4,5],y) is VC. Apply AND: (17,[4,5,11],y) is VSC.
(17,[12,18],13) is VC, (13,[6,7],y) is VC. Apply AND: (17,[6,7,12,13,18],y) is VSC.
(17,[4,5,11],y) is VSC, (17,[6,7,12,13,18],y) is VSC.
Apply OR: (17,[4,5,6,7,11,12,13,18],y) is VC.

(16,[],10) is VC, (10,[],9) is VC. Apply AND: (16,[10],9) is VSC.
(16,[],15) is VC, (15,[],9) is VC. Apply AND: (16,[15],9) is VSC.
(16,[10],9) is VSC, (16,[15],9) is VSC. Apply OR: (16,[10,15],9) is VC.

(9,[],2) is VC, (2,[],y) is VC. Apply AND: (9,[2],y) is VSC.
(9,[],3) is VC, (3,[],y) is VC. Apply AND: (9,[3],y) is VSC.
(9,[2],y) is VSC, (9,[3],y) is VSC. Apply OR: (9,[2,3],y) is VC.

(16,[],11) is VC, (11,[],12) is VC. Apply AND: (16,[11],12) is VSC.
(16,[],17) is VC, (17,[],12) is VC. Apply AND: (16,[17],12) is VSC.
(16,[11],12) is VSC, (16,[17],12) is VSC. Apply OR: (16,[11,17],12) is VC.

(12,[],5) is VC, (5,[],y) is VC. Apply AND: (12,[5],y) is VSC.
(12,[], 6) is VC, (6,[],y) is VC. Apply AND: (12,[6],y) is VSC.
(12,[5],y) is VSC, (12,[6],y) is VSC. Apply OR: (12,[5,6],y) is VC.

(16,[10,15],9) is VC, (9,[2,3],y) is VC. Apply AND: (16,[2,3,9,10,15],y) is VSC.
(16,[11,17],12) is VC, (12,[5,6],y) is VC. Apply AND: (16,[5,6,11,12,17],y) is VSC.
(16,[2,3,9,10,15],y) is VSC, (16,[5,6,11,12,17],y) is VSC.
Apply OR: (16,[2,3,5,6,9,10,11,12,15,17],y) is VC.

(x,[16,19],15) is VC, (15,[1,2,3,4,8,9,10,14],y) is VC.
Apply AND: (x,[1,2,3,4,8,9,10,14,15,16,19],y) is VSC.
(x,[],17) is VC, (17,[4,5,6,7,11,12,13,18],y) is VC.
Apply AND: (x,[4,5,6,7,11,12,13,17,18],y) is VSC.
(x,[],16) is VC, (16,[2,3,5,6,9,10,11,12,15,17],y) is VC.
Apply AND: (x,[2,3,5,6,9,10,11,12,15,16,17],y) is VSC.
(x,[1,2,3,4,8,9,14,15,16,19],y) is VSC.
(x,[4,5,6,7,11,12,13,17,18],y) is VSC.
(x,[2,3,5,6,9,10,11,12,15,16,17],y) is VSC.
Apply OR: (x,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19],y) is VSC.
QED

The "ladder"

The following sequence of deductions proves this virtual connection.

(3,[],a) is VC, (a,[],1) is VC. Apply AND: (3,[],1) is VC.

(3,[],1) is VC, (1,[],y) is VC. Apply AND: (3,[1],y) is VSC.
(3,[],2) is VC, (2,[],y) is VC. Apply AND: (3,[2],y) is VSC.
(3,[1],y) is VSC, (3,[2],y) is VSC. Apply OR: (3,[1,2],y) is VC.

(5,[],3) is VC, (3,[1,2],y) is VC. Apply AND: (5,[1,2,3],y) is VSC.
(5,[],4) is VC, (4,[],y) is VC. Apply AND: (5,[4],y) is VSC.
(5,[1,2,3],y) is VSC, (5,[4],y) is VSC. Apply OR: (5,[1,2,3,4],y) is VC.

(7,[],5) is VC, (5,[1,2,3,4,],y) is VC. Apply AND: (7,[1,2,3,4,5],y) is VSC.
(7,[],6) is VC, (6,[],y) is VC. Apply AND: (7,[6],y) is VSC.
(7,[1,2,3,4,5],y) is VSC, (7,[6],y) is VSC. Apply OR: (7,[1,2,3,4,5,6],y) is VC.

(9,[],7) is VC, (7,[1,2,3,4,5,6],y) is VC. Apply AND: (9,[1,2,3,4,5,6,7],y) is VSC.
(9,[],8) is VC, (8,[],y) is VC. Apply AND: (9,[8],y) is VSC.
(9,[1,2,3,4,5,6,7],y) is VSC, (9,[8],y) is VSC. Apply OR: (9,[1,2,3,4,5,6,7,8],y) is VC.

(11,[],9) is VC, (9,[1,2,3,4,5,6,7,8],y) is VC.
Apply AND: (11,[1,2,3,4,5,6,7,8,9],y) is VSC.
(11,[],10) is VC, (10,[],y) is VC. Apply AND: (11,[10],y) is VSC.
(11,[1,2,3,4,5,6,7,8,9],y) is VSC, (11,[10],y) is VSC.
Apply OR: (11,[1,2,3,4,5,6,7,8,9,10],y) is VC.

(13,[],11) is VC, (11,[1,2,3,4,5,6,7,8,9,10],y) is VC.
Apply AND: (13,[1,2,3,4,5,6,7,8,9,10,11],y) is VSC.

(13,[],12) is VC, (12,[],y) is VC. Apply AND: (13,[12],y) is VSC.
(13,[1,2,3,4,5,6,7,8,9,10,11],y) is VSC, (13,[12],y) is VSC.
Apply OR: (13,[1,2,3,4,5,6,7,8,9,10,11,12],y) is VC.

(x,[],13) is VC, (13,[1,2,3,4,5,6,7,8,9,10,11,12],y) is VC.
Apply AND: (x,[1,2,3,4,5,6,7,8,9,10,11,12,13],y) is VSC.
(x,[],14) is VC, (14,[],y) is VC. Apply AND: (x,[14],y) is VSC.
(x,[1,2,3,4,5,6,7,8,9,10,11,11,12,13],y) is VSC, (x,[14],y) is VSC.
Apply OR: (x,[1,2,3,4,5,6,7,8,9,10,11,12,13,14],y) is VC.
QED

Appendix C

Below we analyze how H-search works for two simple examples of virtual connections presented in Fig. C.1. Diagram 1 is a chain of n two bridges and Diagram 2 is a binary tree with n levels . Both virtual connections are shown with $n = 4$.

Both virtual connections have depth $D = 2n$, and both of them can be proven by H-search. The initial number of empty cells N is equal to $2n$ for the chain of two-bridges and $2(2^n - 1)$ for the binary tree.

Let T_n be the number of the AND and OR deduction rules, which H-search must apply in order to discover the virtual connections. We show that for both of them holds:

$$T_n = \mathrm{O}\big(N^c\big), \tag{C.1}$$

where N is the number of cells in their carriers, and c stands for a positive constant independent of n.

We characterize each cell of the chain of two bridges (see Fig. C.1) by its *elevation*, such that the lowest cell y has the elevation 0 and the highest cell x has the elevation $4n$. Then we characterize each virtual connection and semi-connection belonging to this game with

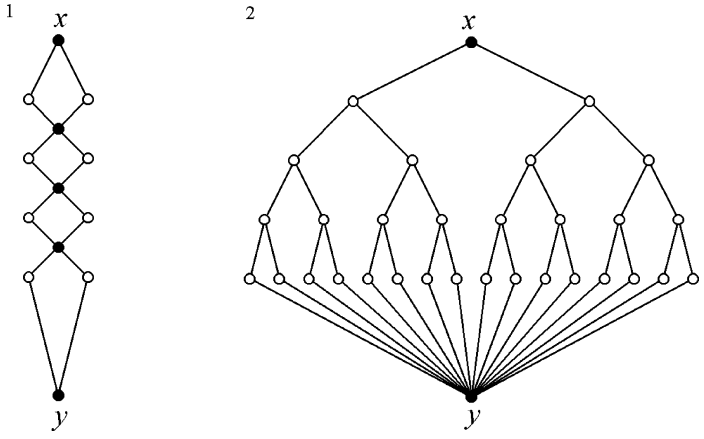

Fig. C.1. Diagrams of virtual connections.

its *height*, equal to the difference of elevations between the higher and the lower ends of the virtual connection or semi-connection. The estimation (C.1) follows from a sequence of simple observations.

(1) The heights of virtual connections and semi-connections produced by the ith iteration of H-search do not exceed $h_i = 2^i$.
(2) Since the heights of the virtual connections and semi-connections produced by the ith iteration of H-search do not exceed h_i, their number does not exceed $v_i = c_1 n h_i$, where c_1 is a constant independent of n.
(3) In case of a chain of two-bridges, H-search does not build more than two semi-connections with the same ends. Hence, the number of the AND and OR deduction rules applied by the $(i+1)$th iteration of H-search does not exceed $d_i = v_i^2$.
(4) It follows from the observations (1)–(3) that the number of AND and OR deduction rules applied by the first k iterations of H-search does not exceed $c_2 n^2 2^{2k}$ for large k, where c_2 is a constant independent of n.
(5) The maximal height of the newly-discovered virtual connections doubles with each iteration. Hence, the number of iterations k that are necessary to build a winning virtual connection does not exceed $\log_2(2n)$.
(6) Inequality (C.1) follows from observations (4) and (5).

Now we estimate the number T_n for the binary tree. Two children of the root x (see Fig. C.1) are the roots of two binary trees with $n-1$ levels. Hence, the following inequality is true:

$$T_n \leqslant 2T_{n-1} + c_3,$$

where c_3 is a constant independent of n. Applying the last inequality repeatedly, we obtain

$$T_n \leqslant 2^n (T_0 + c_3),$$

which implies (C.1).

References

[1] G. Adelson-Velsky, V. Arlazarov, M. Donskoy, Algorithms for Games, Springer, Berlin, 1988.
[2] V.V. Anshelevich, The Game of Hex: The Hierarchical Approach, Combinatorial Game Theory Workshop, MSRI, Berkeley, July 2000, http://www.msri.org/publications/ln/msri/2000/gametheory/anshelevich/1/index.html.
[3] V.V. Anshelevich, The game of Hex: An automatic theorem proving approach to game programming, in: Proc. AAAI-2000, Austin, TX, AAAI Press, Menlo Park, CA/The MIT Press, Cambridge, MA, 2000, pp. 189–194.
[4] V.V. Anshelevich, Hexy wins Hex tournament, ICGA J. 23 (3) (2000) 181–184.
[5] E.R. Berlekamp, J.H. Conway, R.K. Guy, Winning Ways for your Mathematical Plays, Academic Press, New York, 1982; 2nd Edition, A K Peters, Natick, MA, 2001.
[6] E.R. Berlekamp, D. Wolfe, Mathematical Go: Chilling Gets the Last Point, A K Peters, Wellesley, MA, 1994.
[7] C. Browne, Hex Strategy: Making the Right Connections, A K Peters, Natick, MA, 2000.
[8] M. Campbell, A.J. Hoane Jr., F-h. Hsu, Deep blue, Artificial Intelligence 134 (2002) 57–83 (this issue).
[9] J.H. Conway, On Numbers and Games, Academic Press, London, 1976; 2nd Edition, A K Peters, Natick, MA, 2001.

[10] S. Even, R.E. Tarjan, A combinatorial problem which is complete in polynomial space, J. Assoc. Comput. Mach. 23 (4) (1976) 710–719.
[11] D. Gale, The game of Hex and the Brouwer fixed-point theorem, Amer. Math. Monthly 86 (1979) 818–827.
[12] M. Gardner, The Scientific American Book of Mathematical Puzzles and Diversions, Simon and Schuster, New York, 1959.
[13] M. Gardner, The Second Scientific American Book of Mathematical Puzzles and Diversions, Simon and Schuster, New York, 1961.
[14] A.D. de Groot, Thought and Choice in Chess, Mouton Publishers, The Hague, Netherlands, 1965.
[15] D.N.L. Levy, D. Broughton, M. Taylor, The SEX algorithm in Computer Chess, ICCA J. 12 (1) (1989) 10–21.
[16] M. Müller, Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 578–583.
[17] S. Reisch, Hex ist PSPACE-vollständig, Acta Informatica 15 (1981) 167–191.
[18] J.A. Robinson, An overview of mechanical theorem proving, in: R. Banerji, M. Mesarovic (Eds.), Theoretical Approaches to Non-Numerical Problem Solving, Springer, New York, 1970, pp. 2–20.
[19] J. Schaeffer, R. Lake, P. Lu, M. Bryant, Chinook: The World Man-Machine Checkers Champion, AI Magazine 17 (1) (1996) 21–29.
[20] C.E. Shannon, Computers and automata, Proceedings of Institute of Radio Engineers 41 (1953) 1234–1241.
[21] G. Strang, Linear Algebra and Its Applications, Academic Press, New York, 1976.
[22] J. van Rijswijck, Are Bees better than Fruitflies? (Experiments with a Hex Playing Program), in: H. Hamilton (Ed.), AI'00: Advances in Artificial Intelligence, 13th Biennial Canadian Society for Computational Studies of Intelligence (CSCSI) Conference, Springer, New York, 2000, pp. 13–25.